Esercizio di Sincronizzazione Tra Processi: Utenti con Proprio Peso

Testo:

Un ponte, ovviamente stretto, consente il passaggio ad utenti in un solo verso di percorrenza alla volta. Inoltre impone un limite fissato di capacità, CAPACITA, oltre il quale non possono essere ammessi più utenti.

Il traffico si presenta nei due versi e solo uno dei due e' quello ammesso ad entrare sul ponte in un certo istante.
Si vuole una soluzione che:
A) non blocchi l'accesso in nessun caso a ponte scarico;
B) non privilegi una delle due direzioni, in caso di traffico consistente in entrambi i versi.
Ogni utente e' il solo a conoscere il proprio peso (per esempio grazie ad una funzione miopeso). Se il carico corrente più il peso dell'utente supera la CAPACITA, l'utente non può accedere.

Si discuta la politica di sincronizzazione che si vuole realizzare e poi la si traduca secondo uno o più dei costrutti: monitor, regione critica condizionale, rendez-vous di ADA.

Soluzione con il costrutto Monitor:

Per evitare la starvation, si introduce un numero massimo di passaggi in effetto quando il ponte è con traffico da entrambi i versi (PASSAGGI).

```
program PonteConCapacita (input, output);

const  CAPACITA = ....; {miopeso è sempre inferiore per ogni utente}
       PASSAGGI  = ...;

type d = (nord, sud);
    { direzione }

type ponte = monitor;
var cont : integer; { contatore degli utenti che hanno acquisito il ponte }
    dir  :  d;        { direzione corrente del ponte}
    code : array [d] condition; { coda di attesa per i processi}
    npass:integer;{ contatore dei passaggi nel verso corrente}
    capac: integer  { capacità correntemente supportata dal ponte}

procedure entry IN (miadir : d, miopeso : integer);
{importante: notiamo che poiché è solo l'utente che conosce il proprio peso sarà lui il solo a poter
controllare se è nelle condizioni di accedere o meno. Per questo qui non può bastare un if nel
controllo delle condizioni: è necessario un while. Questo comporta poi che chi va a scegliere vada a
svegliare controllando di non rischiare di entrare in un ciclo di risveglio infinito. Nel nostro caso
questo non succede perché adottiamo la tecnica di risveglio a catena }
begin  while   (cont<>0 and dir <> miadir) or
            ((capac + miopeso ) > CAPACITA) or
            ((dir = miadir) and (npass >= PASSAGGI)
               and code [other (miadir)].queue)        do
                        { sospensione se ci sono processi in direzione opposta o se si eccede la
                                    capacità o si è raggiunto il numero massimo di passaggi }
                                       code [miadir]  .wait;
      if dir <> miadir then npass := 0;
                    cont + := 1; dir := miadir; npass + := 1;
                    capac + := miopeso;
                    code [miadir].signal;
end; {IN}
{se il ponte è vuoto non ci si sospende, indipendentemente dalla direzione (A). Se c'è sempre
traffico, npass blocca oltre un certo passaggio}
{ si segnala il processo accodato dopo, in quanto ogni segnalante   in OUT segnala solo il primo. Il
primo processo che viene bloccato, per questione di peso, non da' l'avvio ad altri successivi: si
mantiene quindi l'ordine di arrivo dei processi. In alternativa, si sarebbe potuto tenere un contatore
dei processi sospesi, e, in OUT, segnalarli tutti. Anche un processo di grande peso, lascerebbe
quindi il passo ad altri arrivati dopo ma più leggeri. Problema di starvation}
```

```
procedure entry OUT ( miadir: d, miopeso: integer);
begin
                cont := cont - 1; capac := capac - miopeso;
                if cont = 0 then
                begin
                   npass := 0;
                   if coda [other(dir)]. queue then code [other(dir)].signal;
                     {segnala il primo che segnala il secondo etc. }
                   else code [miadir].signal;
                            {segnala i processi nella propria direzione}
                end;
                else code [miadir]. signal;

end; {OUT}

function other (dir: d): d;
begin if dir = nord then other:=sud else other:=nord end;

begin cont := 0; dir := nord; npass :=0; capac := 0; end;

type utenti = process;
var dir : d;
begin
repeat
  < scegli dir>
          PASS.IN (dir, miopeso);
          <passaggio sul ponte>
          PASS.OUT (dir, miopeso);
until false;
end;
var PASS: ponte;
   A1, A2, ...  ,An: utenti;
begin end.
```

Con l'uso delle regioni critiche condizionali:

```
VAR ponte : shared record
    cont : integer; { contatore degli utenti che hanno acquisito il ponte }
    dir  :  d;        { direzione corrente del ponte}
    npass:integer;{ contatore dei passaggi nel verso corrente}
    req : array [d] of integer;
          end record;


procedure in (miadir : d);
begin
 { si entra nella regione, prima per prenotarsi, poi per accedere realmente }
region ponte do
    req [miadir] := req [miadir] + 1;
end;
region ponte when ((((miadir = dir) and (npass < PASSAGGI)) or
      ((miadir = dir) and (npass >= PASSAGGI)
                  and (req[other (miadir)]=0)) or
      ((miadir<>dir)and(cont=0))
                   and(req[other(miadir)] = 0 ))   and
           ((capac + miopeso) < CAPACITA)  )
{si entra se o
          nella direzione corretta e entro PASSAGGI
          nella direzine corretta, oltre PASSAGGI ma se non ci sono  richieste nell'altro verso
          nella direzione opposta ne' utenti ne' richieste  }
do
  if dir <> miadir then npass := 0;
  dir := miadir; cont + := 1;
  capac  + := miopeso; npass + := 1;  req [miadir] - := 1;
end;
end in;


procedure out (miadir : d);
begin
region ponte
do
  cont - := 1;
  capac  - := miopeso;
  if ((npass >= PASSAGGI) and (cont = 0)) then
    npass := 0;
    if (req [other (miadir)] <> 0) then dir := other (dir)
end out;
```

{La soluzione quindi tiene conto della specifica A) e B).
Se il ponte e' vuoto e non ci sono richieste, il primo che arriva puo' sempre accedere al ponte indipendentemente dal verso corrente, che viene imposto dal processo.
Se c'e traffico, quindi richieste, queste bloccano a PASSAGGI il numero di passaggi consecutivi. }

I processi sono

```
process type utente;
var dir : d;
begin
  loop   <scegli dir>
                    in (dir);
                    <passa sul ponte>
                    out(dir);
         end loop;
end utente;

var u1, ..., un : utente;
```

Soluzione con l'uso di ADA:

In ADA la cosa puo' rispettare le specifiche solo se riusciamo a tenere conto e delle richieste gia' arrivate (usando una procedure entry di prenotazione), e se poi teniamo conto della possibilita' di rifiutare l'accesso per via del peso:

```
package applicazione is
task ponte is
  entry request (miopeso: in integer; miadir: in d; b: out boolean);
  entry inN;
  entry inS;
  entry out (miopeso: in integer; miadir: in d);
end ponte;

task body ponte is
    cont : integer; { contatore degli utenti che hanno acquisito                    il ponte }
    dir  :  d;        { direzione corrente del ponte}
    npass:integer;{ contatore dei passaggi nel verso                    corrente}
    req : array [d] of integer;
          capac : array [d] of integer;
begin
 cont := 0; dir := nord; npass :=0;
          capac [nord] := 0; capac [sud] := 0;
 loop
    select
      accept
                    request (miopeso:in integer; miadir :in d; b:out boolean) do
          if(capac[miadir] + miopeso > CAPACITA)  then
                                  return false
                    else req [miadir] + := 1;
                            capac [miadir] + := miopeso;
                            return true;
                    end request;
{ teniamo conto del peso all'atto della richiesta: quindi i processi utenti devono ripetere la richiesta fin a che questa viene accettata }
  or
                    when (((nord = dir) and (npass < PASSAGGI)) or
          ((nord = dir) and (npass = PASSAGGI) and (req[sud]=0)) or
          ((nord <> dir) and (cont = 0) and (req[sud] =0)) =>
        accept inN do
                      cont + := 1;
                      if dir <> nord then dir := nord; npass := 0; end if;
                            npass + := 1; req [nord] - := 1;
                    end inN;
      or
                    when (((sud = dir) and (npass < PASSAGGI)) or
          ((sud = dir) and (npass = PASSAGGI) and (req[nord]=0)) or
          ((sud <> dir) and (cont = 0) and (req[nord]=0)) =>
        accept inS do
                      cont + := 1;
                      if dir= nord then dir := sud; npass := 0; end if;
```

```
                    npass + := 1; req[sud] - := 1;
                end inS;  {si cambia verso solo se non ci sono richieste nel verso    opposto,
               questo per le in. In caso di out, dopo un certo    numero di passaggi invece si
               impone il verso opposto }
        or
      accept out (miopeso: in integer; miadir: in d) do
                  cont - := 1;
                      capac  - := miopeso;
  if ((npass = PASSAGGI) and (cont = 0)) then
     npass := 0;
     if (req [other (miadir)] <> 0) then
                          dir := other (dir);
                       end if;
                      end if;
                end out;
   end select;
end loop;
end ponte;

task type utenteN is ...
end utenteN;
task type utenteS is ...
end utenteS;

task body utenteN is
 miop: integer; dir: d; b: boolean;
begin
  b := false;
  repeat ponte.request (miop, dir, b) until b;
  ponte.inN;   < passa sul ponte >
  ponte.out (miop, dir);
end utenteN;

task body utenteS is
 miop: integer; dir: d; b: boolean;
begin
  b := false;
  repeat ponte.request (miop, dir, b) until b;
  ponte.inS;   < passa sul ponte >
  ponte.out (miop, dir);
end utenteS;

un1, ..., unn: utenteN;
us1, ..., usm: utenteS;
begin
end applicazione.
```